UML for Embedded Real-time Systems and the UML-Extensions by ARTiSAN Software Tools

Andreas Korff

ARTiSAN Software Tools GmbH
Eupener Straße 135-137
D-50933 Köln
Andreas.Korff@artisansw.com

Abstract: Modelling software using the Unified Modelling Language (UML) also for embedded real-time systems (ERS) becomes more and more popular since the complexity of these systems increases as well as the pressure of short time-to-market timescales. These notes will introduce some extensions to the UML notation implemented in the CASE-tool Real-time Studio, their motivation and their integration with the common UML modelling procedures to ensure a complete picture of the embedded system to be developed.

1 Introduction

The UML specifies in its current version 1.4 its intent to visualize, specify, construct and document the artefacts of software-intense systems. The UML is not intended to be a visual programming language. Instead, it is named as „visual modelling language“. So the UML is directly focussed on solving the problems occurring during the development of software – also within ERS:

1. A common understanding of the system requirements for all members of the development team (including the project sponsors). This helps to avoid the biggest thread in developing systems or software: The solution or product does not fit to the (real) requirements.
2. The possibility to present the system (requirements and solutions) definitely and from different perspectives and in different abstraction layers overcomes the possible misunderstandings when using only textual descriptions.
3. The re-use of solution strategies through object orientation: The goal is to develop system components with a well-defined interface to the external world. Instead of decomposing functions in order to handle high complexity we model the responsibilities of subsystems or objects. These are therefore easy to extend or to replace without disturbing the rest of the system.

This points are valid for all software projects. During the development of ERS, there is another problem: The different worlds of systems engineers, hardware and software engineers have to be combined and bridged in a linguistic and conceptual way. Systems

engineers are thinking in abstract solutions, hardware engineers are using integrated circuits and hardware components, and software engineers always express their ideas using algorithms and data or object structures. The quality of an ERS depends on the ability to integrate these “worlds” of thinking. The UML itself heavily is software-minded, so it is rather difficult to express solutions or ideas independent from software or to describe the system’s hardware topology in a detailed way. Additionally, the real-time aspects within the software itself are also difficult to express in plain UML. But it is possible to fill the notational gaps for the modelling of ERS. By doing this, it is important that the UML extensions fit to the UML notation and are usable and readable intuitively.

State-of the-art system (sub-)functions are very complex. But also their interlocking makes it very complicated to project and to implement this functions. The UML approach is helpful here by encapsulating functionality as use cases. The uses cases afterwards can be implemented step by step.

2. An Example System

ARTiSAN offers prospects the opportunity to evaluate the CASE-tool Real-time Studio Professional for a limited time. Within this evaluation, it is useful to show the tool handling , the different diagrams and what is possible to do with a model by some small examples. These examples, however, still are real-time and embedded. One of this examples, a well-known parking lot, is used here to show the typical development of an ERS.

2.1 The Requirements Analysis

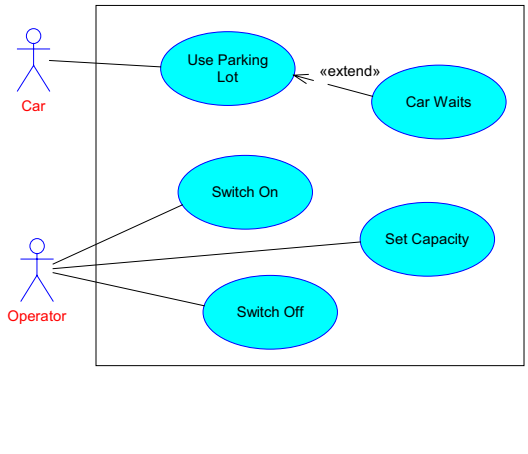

Figure 2.1-1 The Use Case Diagram

Before implementing a system, it is necessary to examine the given requirements in order to avoid gaps or inconsistencies. Within the UML, it is possible to transform functional requirements directly to use cases. In our parking lot example, the identified actors “Car” and „Operator“ are related to the system functionality they are using. The use case diagram in Figure 2.1-1 shows the appropriate relations.

The use case description gives the possibility for the system engineers (and the customer) to describe in a textual way, how the interactions are working between outside (i.e. the actors) and inside the system as well as possible pre- and post-conditions.

Another important aspect of ERS of course refers to the real-time behaviour. In the beginning of requirements modelling they can be expressed as optional property of use cases. Figure 2.1-2 shows this timing note tab for use cases. Because timing is defined as non-functional requirement, this type of information can be collected in the area of non-functional constraints.

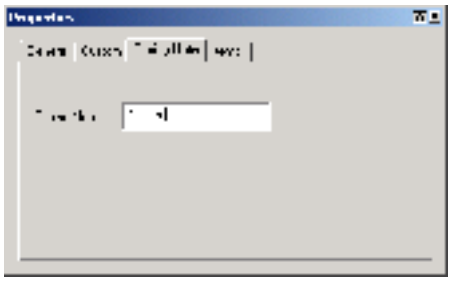

Figure 2.1-1 Use Case Timing Note

Unfortunately the UML does not contain a precise possibility to describe the interfaces at the system boundary in a sufficient way for ERS. Typical for them, the interaction with the outside world does not only happen with „normal“ actors, but also other types of actors like different systems, sensors or generic entities like “the time”. Therefore it is necessary to have a very detailed, but non-software-driven definition of the exact system boundary and of the interfaces the actors can use to interact with the system. In order to support this, ARTiSAN has introduced the context diagram as an UML-Extension within Real-time Studio. In this sub-type of a system architecture diagram, it is possible to show the interface devices related to the specific actors as well as subsystems within the ERS in order to divide the system into functional sub-entities, if necessary. The control system (or software) is drawn as a separate subsystem, thus as black box, connected to the interface devices or other subsystems using communication links for incoming or outgoing messages.

In the parking lot example, we use interface devices to model the sensors realizing that a car wants to enter or to exit the parking lot. Additional devices like push buttons, key pads or displays define the access an operator has to interact with the control system, e.g. to change the number of allowed cars inside the parking lot. Using this type of modelling technique, the use cases for these actors can be modelled in a much more detailed way. Figure 2.1-3 shows the context diagram for the parking lot. The events added onto the information links between the different model elements are used throughout the whole model and their contents are added when available. As an example, there is a “break” event coming from the infrared beam sensor signalling the control system that a car wants to enter the parking lot. This “break” can and will be used in object sequence

diagrams for use case descriptions, in other system architecture diagrams for topological information or in a system state diagram to define the reaction of the system as a whole.

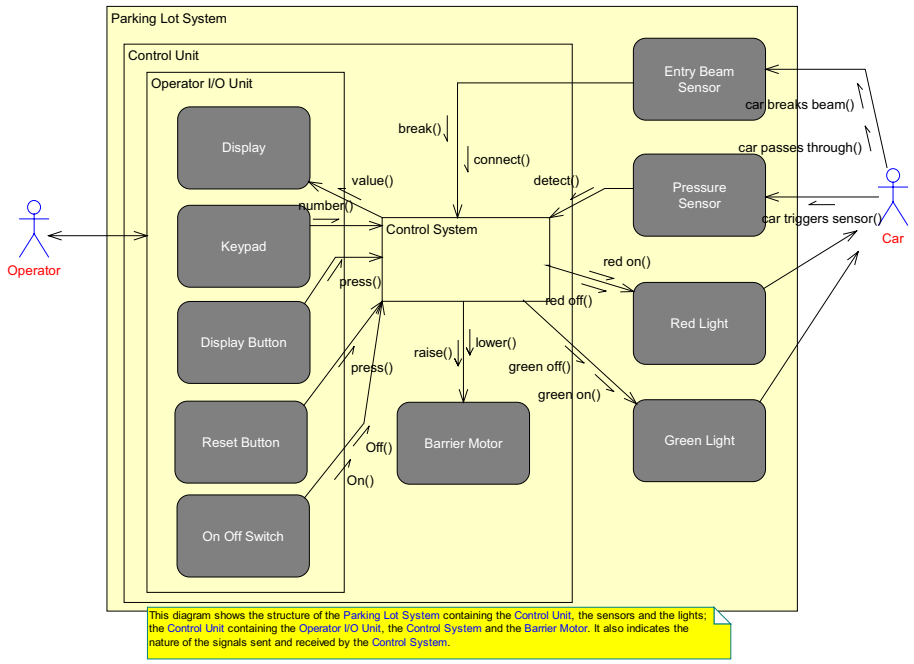

Figure 2.1-3 The Context Diagram

A specific fact that qualifies ERS is the impact of non-functional requirements on the system architecture and the system design. If there are functional limitations, they can be repaired or improved most of the times by additional software updates. Errors or limitations in the area of non-functional requirements like e.g. reliability, timing, size or costs often result in a completely erroneous system architecture, thus resulting in the complete project to fail. In order to repair them, the whole development project must be restarted from the beginning, if possible. Within Real-time Studio, the importance and therefore the necessity to include non-functional constraints into an UML model is covered by the UML extension of Constraints Diagrams. As the name “constraints” implies, this type of requirements have to be linked to the related functional requirements (i.e. use cases), so their range of valid implementations is constrained by the relevant non-functional requirements. In our example, the non-functional constraint “Barrier Motor” defines the time maximum which is allowed for opening or closing the barrier. The use case “Use Parking Lot” is related to this constraint, therefore its design and implementation must follow this timing. Figure 2.1-4 shows the appropriate constraints diagram together with the links editor where constraint and use case can be linked.

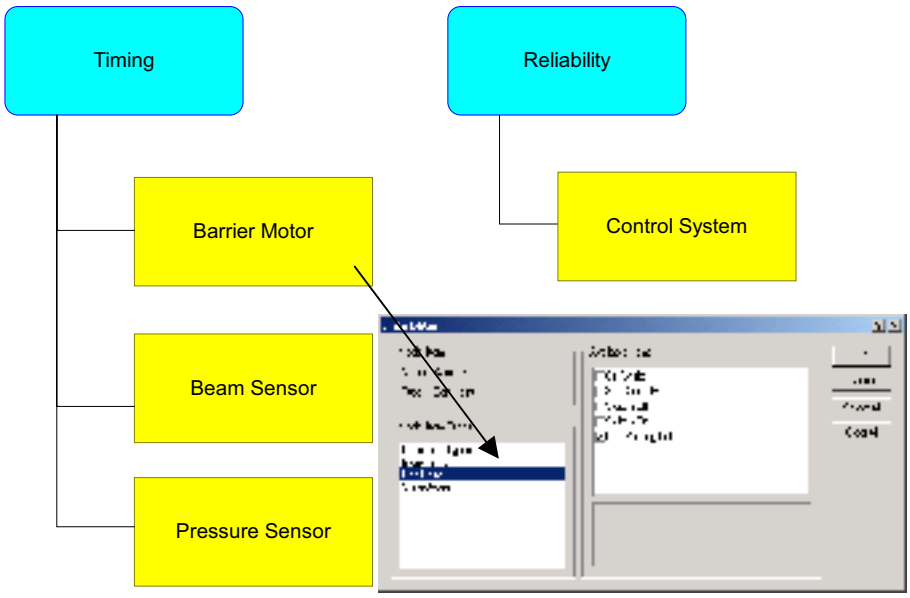

Figure 2.1-4 The Constraints Diagram and possible links to Use Cases

For the requirements analysis of the parking lot model, it is also necessary to formalize the more complex use cases like „Use Parking Lot“. Inside the use case descriptions, the only way to define the possible scenarios is by textual means. An object sequence diagram is a more formal way to describe the use case scenarios, especially taking into account the ARTiSAN extensions for this diagram: Each sequence step is shown in this diagram not only using the object interaction, but also by a textual description. So the transition between the pure textual use case description and a diagram is much easier. Also additional sequence structures can be used inside this textual descriptions like selections, iterations or parallel sequences. This helps to restrict the number of scenarios (or object sequence diagrams) necessary to fully describe the use case. The object sequence diagram itself contains rather more abstract objects like actors, interface devices, subsystems or event. The software control system, which contains the software objects we currently don’t know at this stage, is modelled as black box similar to the context diagram. Two additional graphical items are worth mentioning, too: The system boundary, which was defined already in the context diagram, divides as think line the external actors from the internal system elements like interface devices or software objects. These two types can be differentiated using a dashed line named architectural boundary. This object sequence diagram usage for requirements analysis is shown in Figure 2.1-5.

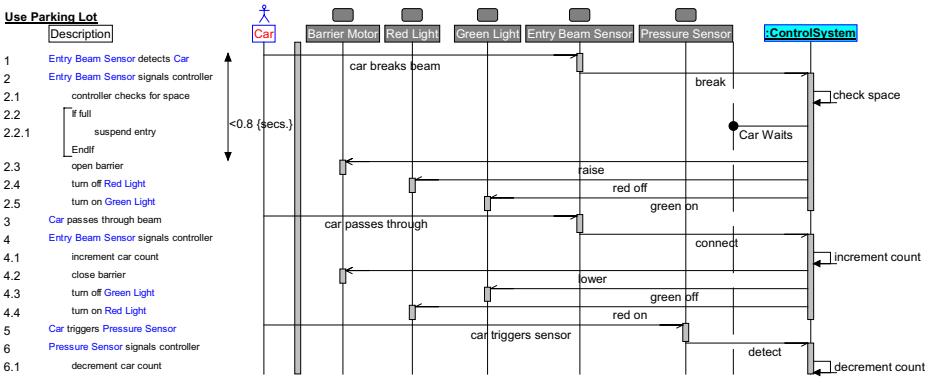

Figure 2.1-5 The Object Sequence Diagram formalizing an Use Case

As a result, the use cases form the basis of the requirements analysis. But to describe the use cases as exact as possible also in their relationships need notational support beyond the “extends” or “includes” typical for UML. For instance is can be necessary to define that the start of a use case is not always possible, some use case scenarios might be mutually exclusive or they might depend on each other. In order to model the overall system behaviour to external events in the granularity level of use cases, a system state diagram can be drawn. The same graphical notation as for state diagrams showing dynamic object behaviour can be used within this diagram. The transition triggers, of course, are the same events running through the system boundary in the context diagram, and the actions within the event action blocks are modelled on use case level.

2.2 The Solution Architecture

In order to create a design model for system implementation, the UML offers a rich set of notations for the object design. Object interaction can be expressed in object collaboration diagrams or object sequence diagrams, for the static object design class diagrams can be used and the dynamic behaviour of objects are modelled in state diagrams. All these features are defined in the UML specification and are located in the object architecture, one of the three abstraction layers in the ARTiSAN incremental, iterative development process “The Real-time Perspective”.

The other two, the system architecture and the software architecture, are specific for ERS, and are described below.

Topological Information about where objects are instantiated in the system, can be expressed in the UML using components diagrams or deployment diagrams. These diagrams cannot reach the depth of detail necessary for ERS. In order to integrate new hardware and software smoothly, additional information like memory-mapping, hardware port addresses, processors and processor types, boards and board types containing board I/O devices, etc. is very helpful.

The interface devices defined in the requirements architecture can now be linked to their appropriate board I/O device with its software specific interface, e.g. a port or register. With this information, the software developer knows the direction to work to very precisely. The common UML model therefore forms the clear communication basis for the development team. Ambiguous ideas, concepts or views, or gaps in the proper definitions can be identified in an early stage and can be solved within the team. For the parking lot example, figure 2.2-1 shows as system architecture diagram the hardware details of the control system:

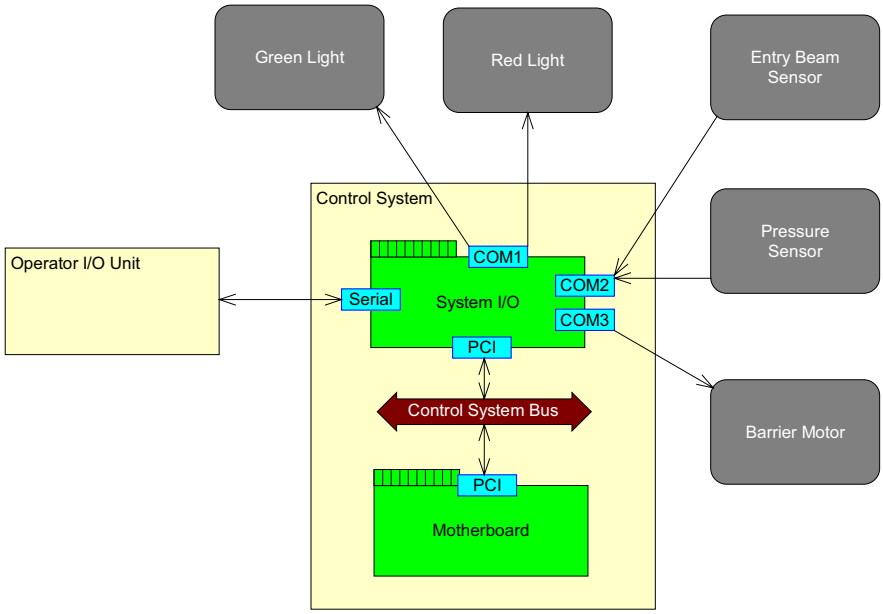

Figure 2.2-1 The System Architecture Diagram

Also on the hardware side, re-use shall be one of the most important design goals. Targeting this, all relevant information must be described using the appropriate aspect. So the same type-instance-model as for the software is used. If e.g. a specific VMEbus board is used in an application, its start address is a property of this one instance of the board. On the other side, the board I/O devices belong to the board type and thus are stored within the board type properties.

For the development of an optimal solution for the real-time aspects of an ERS given in the requirements analysis phase, it is necessary to build up a conception of the relevant concurrent thread or tasks and their communications flow. This concept can be created stand-alone without being mixed with other items of the software object design in order to develop a clear picture of the software architecture. Within Real-time Studio, the UML extension of the concurrency diagram enables the software designer to express his concurrency concepts. Figure 2.2-2 shows the ideas for the parking lot example. From the requirements model, it is clear that both the control of cars entering or exiting the

parking lot and the operator's access to manipulate the number of allowed cars should run in parallel. In order to support fast reaction times for the complete system, a third task is modelled for event detection. This task is connected to the input interface devices like the sensors, the keypad or the discrete pushbuttons. If an external event occurs, the event detection tasks informs the consuming tasks via message queues, event flags, mail boxes or other inter task communication means. These and also the tasks can be linked to the class model within the model item properties. Thus a direct navigation to the relevant class model elements implementing the given concurrency concept is possible.

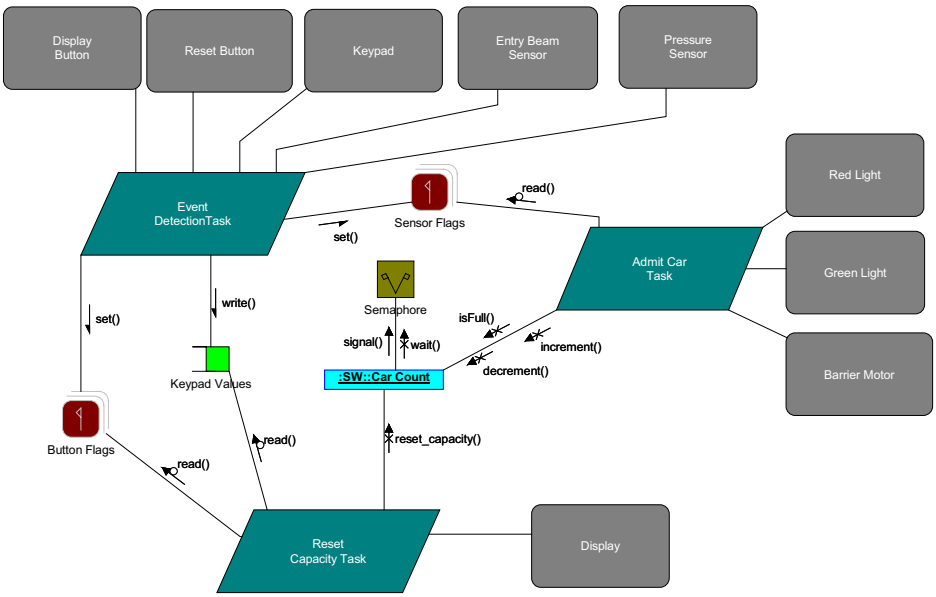

Figure 2.2-2 The Concurrency Diagram

Considering the task sequences themselves, their timing has to be checked against the timing requirements for the events stimulating the task. For this objective, an object sequence diagram can be used as a tasking sequence diagram. All timing information, constraints, timing budgets, calculations and measures are collected here for a given task, thus forming a complete picture by which it is possible to analyse if the requirements can be met. This analysis is supported by the fact, that already every event or message can contain response duration and detection lag timing. The analysis about timing and schedulability can be made either manually or automated by tools. The forthcoming UML Real-time Profile –now in finalization phase in the OMG's Real-time Analysis and Design Working Group will provide a standard set of stereotypes and tagged values. So the modeller can properly define the timing and schedulability constraints in the model, whereas an analysis tool using e.g. the rate-monotonic analysis method can access these data and calculate the corresponding results.

3. Summary

Only with the ability to model information, requirements and solution ideas specific to embedded real-time systems, a complete picture is created for all people involved in the development. Using this, a UML model is created to support the goals of the development project:

- An unambiguous communication basis for all the members of the development team
- A fitting notation for all information layers and –types within the requirements analysis and the solution architecture

All information is stored consistently in a model database. During the development project, the model information is condensed, so an object-oriented solution and implementation can be made up, perfectly matching the given requirements. This is supported by the incremental and iterative development process, which is loosely integrated into the modelling tool. The process can be used as overall development procedure or as on-line help, showing the “red line” through system and software development. If necessary, the process can be adapted according to the relevant company regulations.

In order to use the modelling information, several add-ins are provided. Code synchronizer generate C, C++, Java or Ada Code from class model information, synchronize the differences between code and model level, or reverse-in existing code into UML class model information. The document generator is able to generate according given and adaptable templates the documents fitting the needs of the appropriate project phase. The templates provided out of the box are directly related to the project documents defined in the development process. They can easily be adapted to fulfil the documentation style regulations as well as in the way information is gathered from the model into the documents. There is a model merging tool, and also tools for generating SQL or CORBA IDL statements in order to support distributed systems. Simulation of object interaction up to the simulation of dynamic object behaviour and its link to a graphical prototyping tool complete the tool support to develop the right ERS product the first time.

Bibliography

[OMG02] OMG web site URL: http://www.omg.org/

[Art02] ARTiSAN, Real time Studio Professional®
http://www.artisansw.com/pdflibrary/rtspro_ds_4.pdf

[Art01] ARTiSAN, Real-time studio, Installation Guide, Version 4.1

[MM98] McLaughlin M.J, Moore A., Real-Time Extensions to UML, Dr. Dobb's Journal, December 1998.
http://www.ddj.com/documents/s=913/ddj9812g/9812g.htm

[MC98] Moore A., Cooling N., Real-Time Perspective, Foundation.

http://www.artisansw.com/whitepapers/rtsfoundation.pdf

[MC98_1] Moore A., Cooling N., Real-Time Perspective, Overview

http://www.artisansw.com/whitepapers/rtsoverview.pdf

[Art98] Real-time Perspective Mentor: http://www.artisansw.com/mentor/start.htm

[Art00] ARTiSAN, Model management in Real-time Studio, 2000

[Art01_1] ARTiSAN, Real-time studio®, User's Guide, Version 4.1